Modèles de la théorie des type par traduction de programme

Simon Boulier, Pierre-Marie Pédrot, Nicolas Tabareau

École des Mines de Nantes - INRIA

- Peut-on prouver que $(\forall\, x.\ f\ x = g\ x) \to f = g$?

Modèles par traduction de programme

But : construire un modèle d'une théorie source $\mathcal{S}$

Un terme t de $\mathcal{S}$ $\longrightarrow$ $[t] \in \mathscr{C}$.

Modèles par traduction de programme

But : construire un modèle d'une théorie source $\mathcal{S}$

Un terme t de $\mathcal{S}$ $\longrightarrow$ $[t]$ $\not\in \mathscr{C}$ terme de $\mathcal{T}$.

On compile $\mathcal{S}$ vers $\mathcal{T}$.

Cadre : CC_ω

Termes et types :

- x
- $\square_0$, $\square_1$, $\ldots$
- $\Pi\, x : A.\ B, \quad \lambda\, x : A.\ t, \quad t\ u$
- $\Sigma\, x : A.\ B, \quad (t,\ u), \quad \pi_1\ t, \quad \pi_2\ t$
- $\mathbb{B}$
- $t =_A u$

+ streams, `Prop`, inductifs-récursifs

Nier Funext

Cadre théorique

Autres traductions

- Nier Propext
- Pattern-matching sur $\square$

Nier Funext

Cadre théorique

Autres traductions

- Nier Propext
- Pattern-matching sur $\square$

Nier Funext

But : montrer que $(\Pi\, x : A.\ f\ x = g\ x) \not\to f = g$.

Pour ça on montre que CC_ω + **notFunext** est équiconsistante à CC_ω.

Où **notFunext** axiome de type :

$$\Big(\Pi(A : \square)(B : \square)(f\ g : A \to B).(\Pi\, x : A.\ f\ x = g\ x) \to f = g\Big) \to \bot$$

Nier Funext : Traduction

$\mathrm{CC}_\omega + \textbf{notFunext} \longrightarrow \mathrm{CC}_\omega$

$$
\begin{array}{ll}
[\Pi\, x : A.\ B] & := (\Pi\, x : [A].\ [B]) \times \mathbb{B} \\
[\lambda\, x : A.\ t] & := (\lambda\, x : [A].\ [t],\ \mathtt{true}) \\
[t\ u] & := \pi_1\ [t]\ [u] \\
[\square_i] & := \square_i \\
[x] & := x \\
\ldots & \\
[\textbf{notFunext}] & :=\ (\text{cf. demo})
\end{array}
$$

notFunext :

$$\Big(\Pi(A : \square)(B : \square)(f\ g : A \to B).(\Pi\, x : A.\ f\ x = g\ x) \to f = g\Big) \to \bot$$

Nier Funext : Correction

Théorème

Si $\Gamma \vdash t : A$, *alors* $[\Gamma] \vdash [t] : [A]$.

Nier Funext : Correction

Théorème
Si $\Gamma \vdash t : A$, *alors* $[\Gamma] \vdash [t] : [A]$.

Ex : règle du lambda

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda\, x : A.\ t : \Pi\, x : A.\ B} \qquad \frac{[\Gamma], x : [A] \vdash [t] : [B]}{[\Gamma] \vdash [\lambda\, x : A.\ t] \ :?\ [\Pi\, x : A.\ B]}$$

ok car
$$[\lambda\, x : A.\ t] = (\lambda\, x : [A].\ [t], \mathtt{true})$$
$$[\Pi\, x : A.\ B] = (\Pi\, x : [A].\ [B]) \times \mathbb{B}$$

Nier Funext : Correction

Théorème

Si $\Gamma \vdash t : A$, *alors* $[\Gamma] \vdash [t] : [A]$.

Ex : règle de conversion

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \square \qquad A \equiv B}{\Gamma \vdash t : B}$$

Lemme

Si $t \equiv u$, *alors* $[t] \equiv [u]$.

Nier Funext : Conséquence

Théorème

Si CC_ω *est consistante, alors* $\mathrm{CC}_\omega + \textbf{notFunext}$ *l'est aussi.*

Démonstration.

Nier Funext : Conséquence

Théorème

Si CC_ω *est consistante, alors* $\mathrm{CC}_\omega +$ **notFunext** *l'est aussi.*

Démonstration.

Si $\vdash_{CC_\omega + \textbf{notFunext}} \; t \; : \; \Pi\, X : \Box.\; X.$

Nier Funext : Conséquence

Théorème

Si CC_ω *est consistante, alors* $\mathrm{CC}_\omega +$ **notFunext** *l'est aussi.*

Démonstration.

Si $\vdash_{CC_\omega + \textbf{notFunext}} \; t \; : \; \Pi\, X : \square.\; X.$

Alors $\vdash_{CC_\omega} \; [t] \; : \; (\Pi\, X : \square.\; X) \times \mathbb{B}.$

Nier Funext : Conséquence

Théorème

Si CC_ω *est consistante, alors* $\mathrm{CC}_\omega + \textbf{notFunext}$ *l'est aussi.*

Démonstration.

Si $\vdash_{CC_\omega + \textbf{notFunext}} \; t \; : \; \Pi\, X : \square.\; X.$

Alors $\vdash_{CC_\omega} \; [t] \; : \; (\Pi\, X : \square.\; X) \times \mathbb{B}.$

D'où $\vdash_{CC_\omega} \; \pi_1\; [t] \; : \; \Pi\, X : \square.\; X.$ □

Nier Funext : Plugin

Nier Funext

Cadre théorique

Autres traductions

- Nier Propext
- Pattern-matching sur $\square$

Cadre théorique

$$
\begin{array}{rcl}
\mathcal{S} & \longrightarrow & \mathcal{T} \\
\text{terme } t & \longrightarrow & \text{terme } [t] \\
\text{type } A & \longrightarrow & \text{type } [\![A]\!] \\
\text{contexte } \Gamma & \longrightarrow & \text{contexte } [\![\Gamma]\!]
\end{array}
$$

Où $[\![A]\!] := \iota\ [A]$

et $[\![\Gamma]\!] := x_1 : [\![A_1]\!], \ldots, x_n : [\![A_n]\!].$

Cadre théorique

préservation de la conversion

si $t \equiv u$ alors $[t] \equiv [u]$,

préservation du typage

si $\Gamma \vdash t : A$ alors $[\![\Gamma]\!] \vdash [t] : [\![A]\!]$,

préservation de la consistance

si $[\![\bot_{\mathcal{S}}]\!]$ est habité alors $\bot_{\mathcal{T}}$ l'est aussi.

Théorème

Sous ces conditions, la consistance de $\mathcal{T}$ implique celle de $\mathcal{S}$.

Nier Funext

Cadre théorique

Autres traductions

- Nier Propext
- Pattern-matching sur $\square$

Nier Propext

`Prop` univers imprédicatif :

$$\frac{A : \square_i \qquad x : A \vdash P : \texttt{Prop}}{\prod x : A.\ P : \texttt{Prop}}$$

But : montrer que $(\texttt{P} \leftrightarrow \texttt{Q}) \not\to (\texttt{P} = \texttt{Q})$ pour $\texttt{P}, \texttt{Q} : \texttt{Prop}$.

Nier Propext

$$\mathrm{CC}_\omega + \texttt{Prop} + \textbf{notPropext} \longrightarrow \mathrm{CC}_\omega + \texttt{Prop}$$

$$\begin{aligned}
[\square_i] &:= (\square_i \times \mathbb{B}, \texttt{true}) \\
[\texttt{Prop}] &:= (\texttt{Prop} \times \mathbb{B}, \texttt{true}) \\
[\Pi\, x : A.\ B] &:= (\Pi\, x : [\![A]\!].\ [\![B]\!], \texttt{true}) \\
&\dots \\
[\![A]\!] &:= \pi_1\ [A]
\end{aligned}$$

Rq : on a bien $[\square_i]\ :\ [\![\square_{i+1}]\!]$

car $[\square_i] = (\square_i \times \mathbb{B}, \texttt{true})$
et $[\![\square_{i+1}]\!] = \pi_1\ [\square_{i+1}] = \square_{i+1} \times \mathbb{B}.$

Pattern-matching sur $\square$

$$
\begin{aligned}
&f : \Pi\, A : \square.\ A \to A \\
&\quad f := \lambda(A : \square).\quad \texttt{match}\ A\ \texttt{with} \\
&\qquad\qquad\qquad\qquad\quad \mid \mathbb{B} \Rightarrow \texttt{neg} \\
&\qquad\qquad\qquad\qquad\quad \mid \Pi\, x : B.\ C \Rightarrow \texttt{id} \\
&\qquad\qquad\qquad\qquad\quad \mid\ \square \Rightarrow \texttt{id} \\
&\qquad\qquad\qquad\qquad\quad \texttt{end}
\end{aligned}
$$

$$
\begin{aligned}
f\ \mathbb{B} &\mapsto \texttt{neg} \\
f\ (\mathbb{B} \to \mathbb{B}) &\mapsto \texttt{id}
\end{aligned}
$$

Pattern-matching sur $\square$

$$
\begin{aligned}
&f : \Pi\, A : \square.\ A \to A \\
&\quad f := \lambda(A : \square).\quad \texttt{match}\ A\ \texttt{with} \\
&\qquad\qquad\qquad\qquad\quad \mid \mathbb{B} \Rightarrow \texttt{neg} \\
&\qquad\qquad\qquad\qquad\quad \mid \Pi\, x : B.\ C \Rightarrow \texttt{id} \\
&\qquad\qquad\qquad\qquad\quad \mid\ \square \Rightarrow \texttt{id} \\
&\qquad\qquad\qquad\qquad \texttt{end}
\end{aligned}
$$

$f\ \mathbb{B} \mapsto \texttt{neg}$
$f\ (\mathbb{B} \to \mathbb{B}) \mapsto \texttt{id}$

Idée : traduire $\square$ par un type inductif-récursif sur lequel on peut faire du pattern-matching.

Pattern-matching sur $\square$

```
Inductive TYPE : □ :=
| B  : TYPE
| Pi : Π(a : TYPE). (Elt a → TYPE) → TYPE
| U  : TYPE
with Elt : TYPE → □ := fun
| B      ⇒ 𝔹
| Pi a b ⇒ Π(x : Elt a). Elt (b x)
| U      ⇒ TYPE.
```

Pattern-matching sur $\square$

```
Inductive TYPE : □ :=
| B  : TYPE
| Pi : Π(a : TYPE). (Elt a → TYPE) → TYPE
| U  : TYPE
with Elt : TYPE → □ := fun
| B      ⇒ 𝔹
| Pi a b ⇒ Π(x : Elt a). Elt (b x)
| U      ⇒ TYPE.
```

$$\mathrm{CC}_\omega + \texttt{univ\_rec} \longrightarrow \mathrm{CC}_\omega + \texttt{TYPE}$$

$$\begin{aligned}
[\square] &:= \texttt{U} \\
[\Pi\, x : A.\ B] &:= \texttt{Pi}\ [A]\ (\lambda\, x : [\![A]\!].\ [B]) \\
[\lambda\, x : A.\ t] &:= \lambda\, x : [\![A]\!].\ [t] \\
&\dots \\
[\![A]\!] &:= \texttt{Elt}\ [A]
\end{aligned}$$

Pattern-matching sur $\Box$

Théorème

Si $\mathrm{CC}_\omega + \texttt{TYPE}$ *est consistante, alors* $\mathrm{CC}_\omega + \texttt{univ\_rec}$ *l'est aussi.*

Pattern-matching sur $\Box$

Théorème

Si $\mathrm{CC}_\omega + \texttt{TYPE}$ *est consistante, alors* $\mathrm{CC}^e_\omega + \texttt{univ\_rec}$ *l'est aussi.*

Conclusion

Modèles donnés par traduction de programme $\mathcal{S} \longrightarrow \mathcal{T}$.
Avantages :

- simple
- n'utilise que la théorie des types
- modulaire
- implémentable

4 traductions ($\mathrm{CC}_\omega + \mathit{qqch} \longrightarrow \mathrm{CC}_\omega$) :

- **notFunext**
- **notStreamext**
- **notPropext**

} Formalisées et implémentées comme plugin.

- **univ_rec**

`https://github.com/CoqHott/Program-translations-CC-omega`